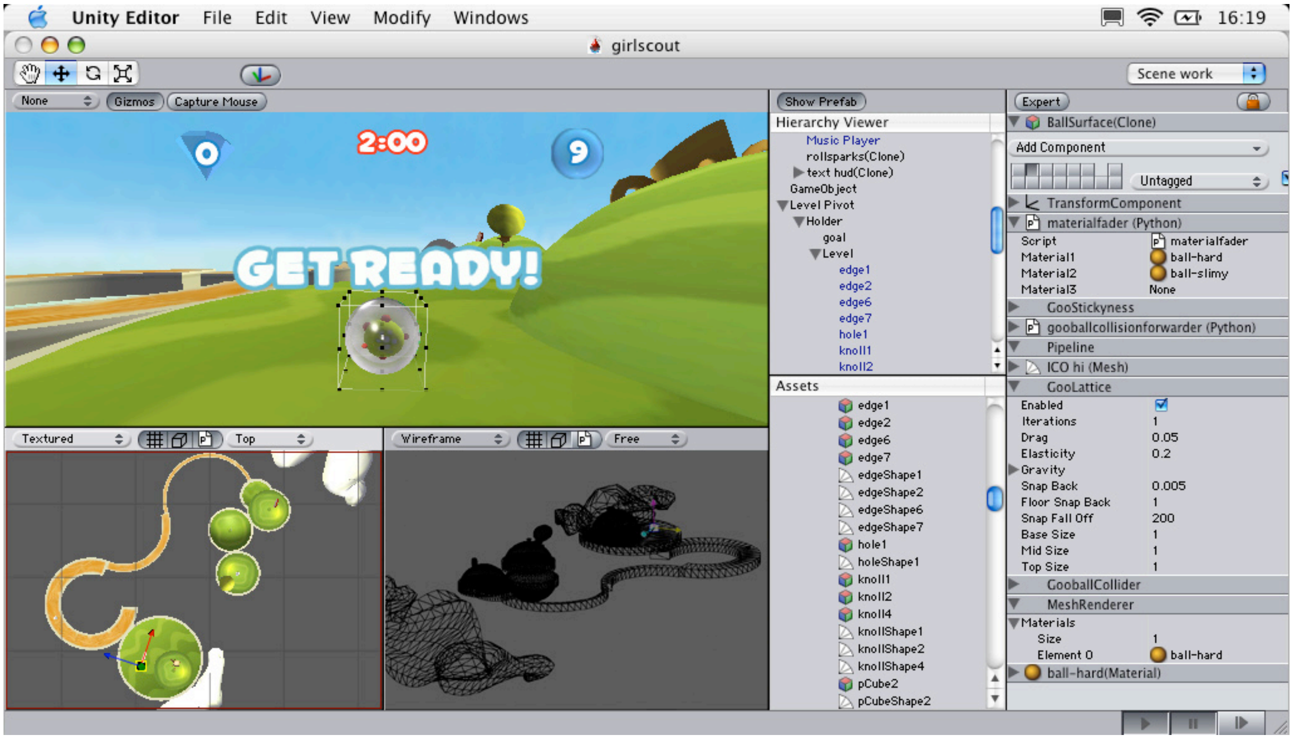
# Unity 1.0



BallSurface(Clone)

TransformComponent

materialfader (Python)

Script materialfader
Material1 ball-hard
Material2 ball-slimy
Material3 None

GooStickyness

gooballcollisionforwarder (Python)

Pipeline

ICO hi (Mesh)

GooLattice

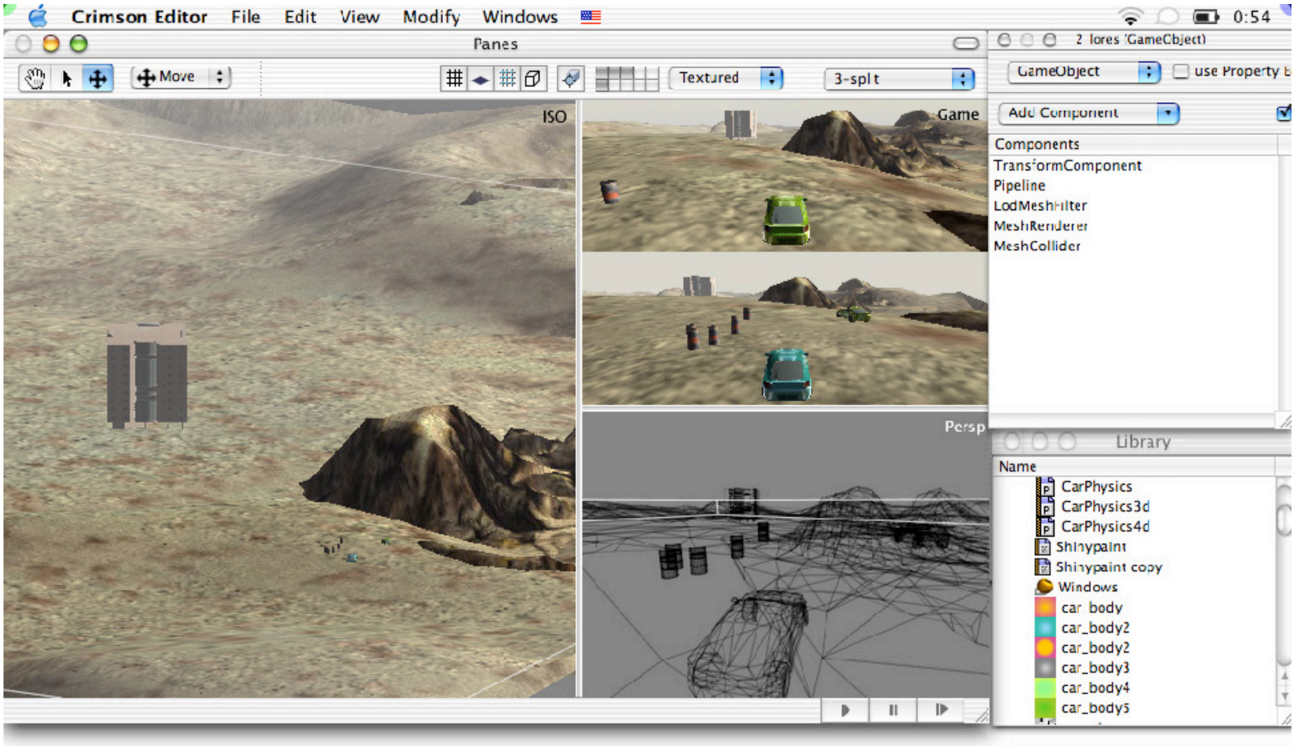
Enabled ☑
Iterations 1
Drag 0.05
Elasticity 0.2
Gravity
Snap Back 0.005
Floor Snap Back 1
Snap Fall Off 200
Base Size 1
Mid Size 1
Top Size 1

GooballCollider

MeshRenderer

Materials
Size 1
Element 0 ball-hard

ball-hard(Material)



# Integrated Editor

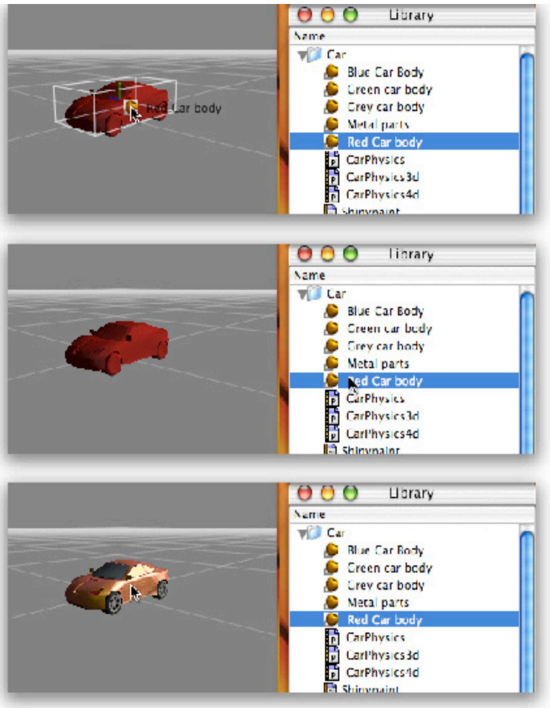
One of the main goals of the Unity Engine is to give the power to experiment to the game de-signers, level de-signers, and game code scripters. The engine features a built-in editor with niceties such as prefabs, undo and copy-paste. It is here that the game is assembled and tested. The editor ties the different parts of the engine together in an intuitive way.

## Rapid Prototyping

While editing, testing a level is never more than a click away. The editor still works while playing, so you can pause the game and examine what is going on.

## Rapid Deployment

When the game is ready, you just select ‘Build Game’ from the main menu and have an instant executable, ready to burn on to a CD for demos or the gold master.



*Full Drag and drop editing – simply drag assets into the scene to use them, then drag materials, behaviours, etc. onto them to change its appearance and function.*

# Data-driven design

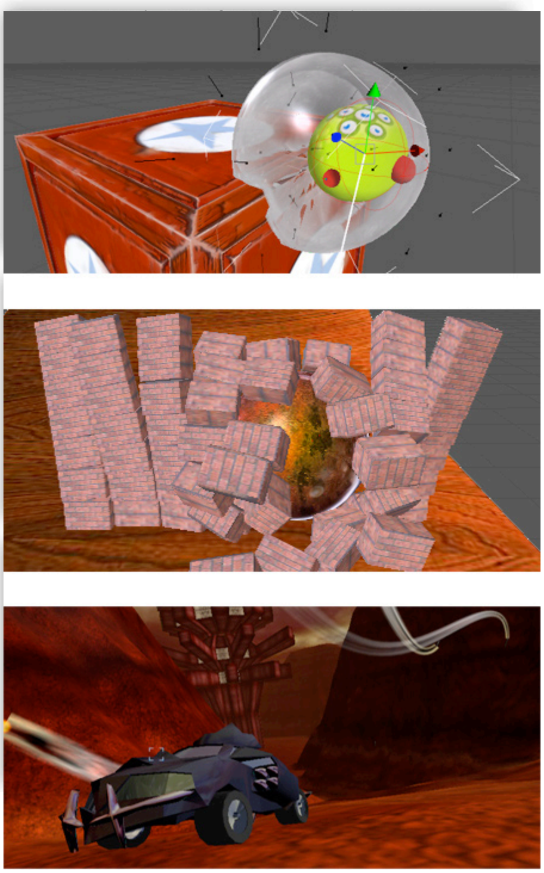
In traditional engines, object behaviours are hard-coded into the engine. This makes it hard to reuse code between projects as well as time consuming to modify the basic behaviours in the last tweaking stages of a project.

## Game Objects

The Unity Engine takes another approach. We have a simple notion of a Game Object, to which you can attach and detach behaviours either in the editor, or while the game is running. This makes it simple to make reusable code, as behaviours no longer are tied to a specific object – or even game.

## Python Scripting

In order to boost efficiency, all apects of a game can be controlled by Python scripts. Python is a simple but powerful programming language that makes it simple and quick to add custom functionality to levels, without spending the time normally associated with adding features.



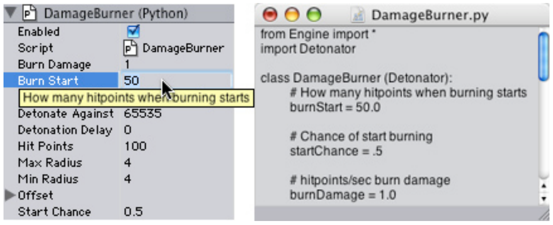
*Flexible physics engine handles everything from gooey balls, through stacked boxes, to dynamic car physics.*

## Prefabs

Once you've made one object, you can turn it into a prefab. This can then quickly be placed throughout the game. This means that level designers can tweak objects without having to repeat the changes to all similar objects.

## Asset Database

At the core of the engine lies the asset database. We read all major file formats and importing is handled transparently. Artists can just save their work in their favourite application and changes are automatically reflected in the game.



| | |
|---|---|
| Enabled | ☑ |
| Script | DamageBurner |
| Burn Damage | 1 |
| Burn Start | 50 |
| Detonate Against | 65535 |
| Detonation Delay | 0 |
| Hit Points | 100 |
| Max Radius | 4 |
| Min Radius | 4 |
| Offset | |
| Start Chance | 0.5 |

How many hitpoints when burning starts

```
from Engine import *
import Detonator

class DamageBurner (Detonator):
	# How many hitpoints when burning starts
	burnStart = 50.0

	# Chance of start burning
	startChance = .5

	# hitpoints/sec burn damage
	burnDamage = 1.0
```

*Python scripts fully participate in physics.*

# Rendering System

The Unity Engine features a highly efficient rendering system, which ties together level-of-detailed geometry and scriptable shaders.

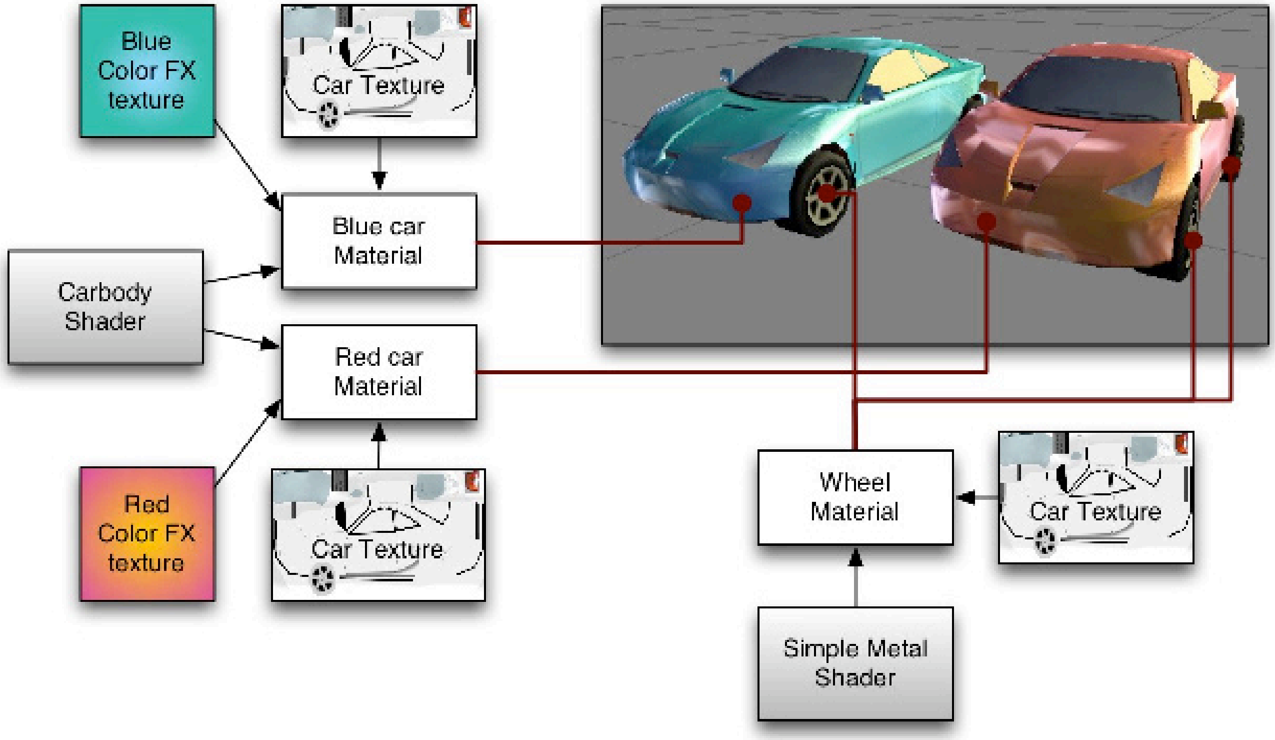
## Scriptable Shaders

Graphics rendering is controlled by easy to create shader scripts that govern how objects appear in-game.
The shader system supports many state-of-the-art shading techniques, enabling creation of fur, water and other highly demanding materials.

## Dynamic Graphics

All graphics in the engine are processed through pipelines – an intuitive way to add modifiers to the objects familiar to anyone who has used 3D Studio Max™.

Dynamic geometry enables artists to modify geometry in the editor, thus greatly speeding up level creation..



*A shader can define any number of editable properties so graphic artists easily can adjust the materials of game assets*

# Feature List

## GameObjects:

- Component based modelling of game entities. Components can be added and removed while the game is running.
- Powerful (and fast) query and messaging system for communication between components.

## Graphics:

- Texture splatting for high detail texturing, can be painted in the editor or generated.
- State of the art quadric error metric based LOD meshes using geomorphing, for optimal speed on modern graphics hardware.
- Clay and magnet tools for manipulating terrain vertices.
- Very fast and flexible particle system which supports collisions as well as color and texture animation.
- Animated textures.
- Metaballs for rendering of fluids.
- Normal map generation from meshes.
- Superb multiple camera support.
- Skyboxes.

## Flexible Shader Engine:

- Support for latest graphics card features such as bumpmapping, reflection mapping, refraction, cubemaps, trilinear filtering and mipmapping.

## Pipelined Graphic Filters:

- Flexible filter architecture where channels can be arbitrarily remapped allowing for a very flexible graphics pipeline.
- Rendering is partitioned into different filters, mesh-filters and mesh-renderers.
- Spline lofts and Lattice deformations
- Modifier filters that tweaks geometry on the fly.

## Animation System:

- Channel based keyframe animation.
- Animation blending & mixing.

## Texture importing:

- Imports all major bitmap and movie formats using QuickTime.
- Heightmap -> normalmap conversion.
- Different mipmap generation methods (Detail fade, normalizes, boxfilter, Kaiser filte & gamma corrected filtering).

## Dynamics system:

- Flexible dynamics system featuring hinges, ball-sockets and suspensions.
- Supports friction, slipping, and restitution.
- Uses LCP system for stable rigid body simulation.
- Box stacking, cars, helicopters, whatever you want to simulate.
- Primitive-Primitve & Mesh-Primitive collisions.

## Serialization:

- High-speed serialization that is simple to implement.
- Serialized variables are edited in a nice property editor.
- Serialization engine is typesafe and can convert values if the type changes.
- Serialization system transparently handles file conversion between different versions.
- In release mode serialization is extremely fast.

## Audio:

- 3D positional audio with Doppler effect, gain and pitch control.
- Support for all common file formats.

## Python scripting:

- Excellent communication between C++ and Python, with all major C++ classes exposed to Python.
- Automatic serialization and property editing in the editor.
- Game logic is most easily implemented in Python.

# System Requirements

## Engine:

PC:
800 Mhz Pentium class processor
Windows 98, 2000 or XP
128 MB RAM recommended

Mac:
600 MHz G3 processor or later
Mac OS X version 10.2.8 or later
256 MB RAM recommended

## Editor:

Requires a PowerMac running OS X panther
At least 384 MB RAM recommended

# Licensing

The Unity Engine is available for 3rd party licensing. Please contact us for more information:

Over The Edge Entertainment
Kronborggade 14, st. th.
DK – 2200 Copenhagen N.
http://www.otee.dk
info@otee.dk